

Discrete Event Simulation Of NASA's Remote Exploration and Experimentation
Project (REE)

Julia Dunphy Ph.D. and Stephen Rogstad
Jet Propulsion Laboratory
California Institute of Technology
4800 Oak Grove Drive
Pasadena
CA 91109 USA
Julia.Dunphy@jpl.nasa.gov
Stephen.Rogstad@jpl.nasa.gov

Background

The Remote Exploration and Experimentation Project (REE) is a new initiative at JPL to be able to place a supercomputer on board a spacecraft and allow large amounts of data reduction and compression to be done before science results are returned to Earth. Since downlink bandwidth is at a premium, this project should lead to a much greater science return from missions to distant parts of the solar system. The requirements for the project include severe restrictions on computer downtime and errors introduced by the harsh environment of space. In particular, the computer must be able to rapidly recover from radiation-induced faults and must also perform internal tests to verify that those faults that do not trigger the recovery system do not result in erroneous data being downlinked. In order to prove the reliability and performance requirements were possible, a fairly significant modeling task was initiated along side the experimental work.

Modeling Methodology

Several university groups along with JPL's own simulation group are involved in this effort. The university groups are working on Markov modeling, automatic workload generation tools and radiation studies. The JPL modeling team is building a large discrete event simulation of a complete 5-year mission using Hy-performix Workbench

as the core and new data representations such as XML for the workload, parameters and results. This short paper only deals with the JPL modeling team, though all the approaches are naturally designed to work together.

Design of the REE

The heart of the REE is the "Core Cluster". The Core Cluster is a set of four computer nodes that ensure the reliability of the array at all times. All decisions internal to the Core Cluster are verified by voting among the four nodes. There is a remote executive (the REX) running on the core cluster that is responsible for interfacing with the main spacecraft computer. The REX accepts commands from the spacecraft and forms application clusters of computing nodes to host the applications as they are accepted. Also resident on the Core Cluster is the health monitor. The health monitor, as its name suggests, keeps track of the health of each computing node and uses this information in deciding which nodes to include in the application clusters as they are formed and disbanded. It is possible for an application to request minimum, maximum and optimal numbers of processing nodes, as available. The application clusters themselves are under the control of an application manager. The application manager looks after the nodes in its cluster by checking for application crashes, hangs, and the production of erroneous data and cluster node

hardware problems. The REE is still in initial design and some of these preliminary ideas are subject to change later.

Design of the Model

The discrete event simulation uses the Hy-performix Workbench tool. The hardware is presently modeling at a fairly high level since we do not yet understand where the bottlenecks are. The applications running on the computer are quite complex and often involve the same types of application components such as fast Fourier transforms, Garbor filters and K-Means. We will be modeling these reusable pieces as time permits to allow the easy construction of more complex calculations.

1. Workload Generation

The classical way to create the workload for the model is to write a file that describes the input transactions. In our case the input transactions are the commands to start up applications, sensor inputs showing that the available power is low and events associated with faults such as radiation events. This file must be parsed, the model transactions generated and their attributes given values. Instead of writing a regular file parser to create the workload, we decided to join the modern age and use XML for the workload format. The workload description is quite well suited to XML and standard parsers such as the SAX parsers from w3.org are easily tailored to read any file that obeys the rules of XML. There are several big advantages in using XML:

1. As already mentioned, it is easy to parse
2. You can look at through a web browser and check for syntactic errors
3. It can be run through a style sheet to produce human readable reports.

While most people think of XML as related to java, the parsers also exist for C and since SES uses C as its language, this is what we used.

2. Model Parameters

Just like the workload, the model parameter file can easily be expressed in XML. To actually be loaded, it must be pre-formatted in the Workload syntax.

3. Fault Modeling

Most faults are due to transient radiation bursts that cause Single Even Upsets (SEUs). Most SEUs will not cause memory errors because of the error correction mechanisms employed. However, sometimes they can occur in such a way that the error correction is ineffective. Also, some caches are not error corrected. These, together with faults occurring in registers are being modeled. When a fault occurs, the error propagation is not trivial to calculate. There are several possible outcomes:

1. The application will crash.
2. The application will hang
3. The application will terminate normally but with the wrong answer
4. The application will terminate normally with correct data but take more time.

We are attempting to get numbers representing the probability of each of these outcomes by measuring them on a test bed.

Status

We have completed our first full five-year scenario run successfully and demonstrated that the system as presently envisioned has a good possibility of meeting the goals set for it.